IN THE CLAIMS:

Please amend the claims as follows:

1. (currently amended) A safety controller comprising:

a processing unit having a processor executing instructions, and a memory holding instructions and data, the processing unit providing a hardware lock preventing writing of at least a portion of the memory as controllable by a lock instruction, wherein the memory is adapted to hold an independently executing a standard program providing control for an industrial process and a safety program and providing control of safety equipment providing human safety in an environment of the industrial process, the safety program ~~requires~~ requiring higher reliability execution than the standard program; and

a lock management program executable on the processing unit determining when the safety program is to be run and based upon this determination, unlocking ~~unlocks~~ a portion of memory holding the safety program at times when the safety program is ~~executing~~ to run and locking the portion of memory at other times when the safety program is finished running and the standard program is to be run so that the standard program when running cannot corrupt the safety program.

2. (original) The safety controller of claim 1 wherein the portion of memory also holds data operated on by the safety program.

3. (original) The safety controller of claim 2 further including I/O circuitry exchanging input/output values with an external machine and wherein the data includes input/output values.

4. (original) The safety controller of claim 1 wherein the lock management program executable on the processing unit is different from the safety program.

5. (original) The safety controller of claim 4 wherein the lock management program executable on the processing unit is an operating system running on the processing unit and scheduling the execution of the safety program and standard program.

6. (original) The safety controller of claim 1 wherein the lock management program executable on the processing unit confirms the memory portion is locked at the start of the safety program before unlocking the memory portion and invokes an error if the memory portion is not locked at the start of the safety program before unlocking the memory portion.

7. (currently amended) The safety controller of claim 1 wherein lock instruction is a setting of a register that may be read by the ~~locking~~ lock management program indicating the status of different memory portions as locked and unlocked.

8. (original) The safety controller of claim 1 wherein the hardware lock operates so that the locked portion of memory may be read.

9. (original) The safety controller of claim 1 wherein the hardware lock operates so that different portions of memory may be simultaneously locked and unlocked.

10. (original) The safety controller of claim 1 wherein the lock management program executes to keep the portion of memory holding the standard program unlocked.

11. (original) The safety controller of claim 1 wherein the lock management program is a portion of the safety program unlocking the memory portion at the start of safety program and locking the memory portion at the conclusion of the safety program.

12. (original) The safety controller of claim 1 wherein the portion of memory holding the safety program also holds a copy of selected data generated by the standard program.

13. (previously presented) The safety controller of claim 1 further including a lock check program periodically checking the status of the portion of memory holding the safety program

when a safety program is not executing and invoking an error is- if the memory portion holding the safety program is unlocked.

14. (original) The safety controller of claim 1 further including:

a second processing unit having a processor executing instructions, and a memory adapted to hold a copy of the safety program; and

a synchronization program executable by the processing units to execute the safety program on both processing units and compare execution of the safety programs and to enter a safety state when this execution differs.

15. (original) The safety controller of claim 14 wherein the second processing unit provides a hardware lock preventing writing of at least a portion of the memory adapted to hold a copy of the safety program as controllable by a lock instruction.

16. (currently amended) A method of operating a safety controller having a processing unit with a processor executing instructions, and a memory holding instructions and data, the processing unit providing a hardware lock preventing writing of at least a portion of the memory as controllable by a lock instruction, the method comprising the steps of:

(a) loading a first portion of memory with a standard program providing control for an industrial process and a second portion of memory with a safety program providing control of safety equipment and providing human safety in an environment of the industrial process, the safety program requiring higher reliability execution than the standard program;

(b) executing the safety program and standard program at different times and determining when the safety program is to be run and based upon this determination, unlocking the second portion of memory at times when the safety program is ~~executing~~ to run and locking the second portion of memory at other times when the safety program is finished running and the standard program is to be run so that the standard program when running cannot corrupt the safety program..

17. (original) The method of claim 16 wherein the second portion of memory also holds data operated on by the safety program.

18. (original) The method of claim 17 further wherein the safety controller includes I/O circuitry exchanging input/output values with an external machine and wherein the data includes input/output values.

19. (original) The method of claim 16 further including the step of confirming the second portion of memory is locked at the start of the safety program before unlocking the second portion of memory and invoking an error if the second portion of the memory portion is not locked before the unlocking.

20. (original) The method of claim 16 wherein the lock instruction is a setting of a register that may be read by a program indicating the status of different memory portions as locked and unlocked.

21. (original) The method of claim 16 wherein locked memory may be read but not written to.

22. (original) The method of claim 16 wherein different portions of memory are simultaneously locked and unlocked.

23. (original) The method of claim 16 wherein the first portion of memory remains unlocked.

24. (original) The method of claim 16 further including the step of periodically checking the status of the second portion of memory when a safety control program is not executing and invoking an error if the memory portion is unlocked.

25. (original) The method of claim 16 further wherein the safety controller includes a second processing unit having a processor executing instructions, and a memory adapted to hold a copy of the safety program, and including the step of: executing the safety program on both processing units and comparing execution of the safety programs to enter a safety state when this execution differs.

26. (currently amended) A method of operating a safety controller system comprising the steps of:

(a) accepting program instructions from a user describing the logical combination of input sensor data to produce output control data;

(b) collecting the program instructions into logical tasks;

(c) identifying the task as to one of two levels of reliability, a first level being of higher reliability than the second level;

(d) loading a task of the first level into a first portion of memory and a task of the second level into a second portion of memory;

(e) executing the loaded tasks at different times and determining when the task of the first level is to be run and based upon this determination, unlocking the first portion of memory at times when the task of the first level is executing to run and locking the second portion of memory at other times when the task of the first level is finished running and the task of the second level is to be run so that the task of the second level when running cannot corrupt the task of the first level.